Gary King's Electoral Responsiveness and Partisan Bias Model
8 October 2009

ρ = Electoral Responsiveness

$\lambda_1 , \lambda_2 , \ldots , \lambda_J$ are the Partisan Bias parameters

s_i = Total number of seats available at election i=1, … , n

S_{ij} = Random Variable = number of Seats allocated to party j after election i

s_{ij} = Observed realization of S_{ij} -- Number of seats held by party j after election i

v_{ij} = Vote share party j in election i

Model of Responsiveness and Partisan Bias

Expected Proportion of Seats for Party j = $\dfrac{e^{\lambda_j} v_j^{\rho}}{\sum_{k=1}^{J} e^{\lambda_k} v_k^{\rho}}$

The seat allocation is assumed to be a multinomial:

1) $(S_{i1} , S_{i2} , \ldots , S_{iJi}) \sim \dfrac{s_i!}{s_{i1}!s_{i2}! \ldots s_{iJ_i}!} \pi_{i1}^{s_{i1}} \pi_{i2}^{s_{i2}} \ldots \pi_{iJ_i}^{s_{iJ_i}}$

2) where $\pi_{ij} = \dfrac{E(S_{ij})}{s_i} = \dfrac{e^{\lambda_j} v_j^{\rho}}{\sum_{k=1}^{J_i} e^{\lambda_k} v_k^{\rho}}$

The Likelihood function is:

3) $L(\rho, \lambda_1, \lambda_2, \ldots , \lambda_J | \, s, \, v) = \prod_{i=1}^{n} \dfrac{s_i!}{s_{i1}!s_{i2}! \ldots s_{iJ_i}!} \pi_{i1}^{s_{i1}} \pi_{i2}^{s_{i2}} \ldots \pi_{iJ_i}^{s_{iJ_i}}$

The natural log of the Likelihood function is (after dropping constant terms):

4) $\ell nL(\rho, \lambda_1, \lambda_2, \ldots , \lambda_J | \, s, \, v) = \sum_{i=1}^{n} \sum_{j=1}^{J_i} s_{ij} \left[\lambda_j + \rho \ell n(v_{ij}) - \ell n \left(\sum_{k=1}^{J_i} e^{\lambda_k + \rho \ell n v_{ik}} \right) \right]$

The Posterior distribution is:

5) $\xi(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu) \propto L(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu)\xi(\rho)\xi(\lambda_1)\xi(\lambda_2)\ldots\xi(\lambda_J)$

Taking the natural log of the Posterior:

$\ell n\xi(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu) =$

6) $$\sum_{i=1}^{n}\sum_{j=1}^{J_i} s_{ij}\left[\lambda_j + \rho\ell n(v_{ij}) - \ell n\left(\sum_{k=1}^{J_i} e^{\lambda_k + \rho\ell n v_{ik}}\right)\right] + \ell n\xi(\rho) + \ell n\xi(\lambda_1) + \ell n\xi(\lambda_2) + \ldots + \ell n\xi(\lambda_J)$$

Now, consider the effect of uninformative prior distributions for the parameters. For example, suppose:

$$\xi(\lambda_j) \sim N(0, \sigma^2), \text{ with } \sigma^2 = 1000$$

then

7) $$\ell n\xi(\lambda_j) = -\ell n(\sqrt{2\pi}\sigma) - \frac{\lambda_j^2}{2\sigma^2}$$

Throwing away the constant the real contribution to $\ell n\xi(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu)$ is simply $-\frac{\lambda_j^2}{2\sigma^2}$ which vanishes for large σ^2. Consequently, the standard MLE using $\ell nL(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu)$ is essentially the same as the solution for $\ell n\xi(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu)$. ***Namely, it is easier to simply use a classical equation solver with numerical derivatives*** to solve $\ell n\xi(\rho, \lambda_1, \lambda_2, \ldots, \lambda_J | \text{ s}, \nu)$ rather than Gibbs or Metropolis-Hastings.

KING_ENGLAND_ROYCE.ODC

```
model
      {

      #  PRIORS
        bias[1]<-0
        for (k in 2 : K) { bias[k] ~ dnorm(0,0.00001)} # vague priors
       # Loop around elections:
       rho ~ dexp(1)

       # LIKELIHOOD
        for (i in 1 : I) {     # loop around elections

      # Multinomial model
         X[i,1:K] ~ dmulti( p[i,1:K] , n[i])
         n[i] <- sum(X[i,])
         for (k in 1:K) {     # loop around parties
            p[i,k]        <-  phi[i,k] / sum(phi[i,])
            log(phi[i,k]) <- bias[k] + rho*log(V[i,k])

          }
    }
  }

      }

Data
list( I   = 12,   K   = 3,
      X = structure(.Data = c(
300,  315,  12,
325,  295,  9,
345,  277,  8,
365,  258,  7,
304,  317,  9,
253,  364,  13,
330,  288,  12,
297,  301,  37,
277,  319,  39,
339,  269,  27,
397,  209,  45,
376,  229,  45
), .Dim = c(12, 3)),
V = structure(.Data = c(
.4341994,   .4610934,   .1047072,
.4797144,   .4877813,   .0325043,
.4974225,   .4635792,   .0389983,
.4935235,   .4384425,   .0680339,
.4339797,   .4413256,   .1246946,
.4187992,   .4803503,   .1008505,
.4637579,   .430723,    .1055191,
.3788168,   .3715876,   .2495956,
.3584408,   .3925122,   .249047,
.4387356,   .3693695,   .1918949,
.4242528,   .2757294,   .3000178,
.423, .308, .269), .Dim = c(12, 3))
```

```
)


Inits

list(bias = c(NA, 0, 0),
          rho =  0)
```

KING_ENGLAND_ROYCE_2.ODC

```
model
{

      #  PRIORS
        bias[1]<-0
        for (k in 2 : K) { bias[k] ~ dnorm(0,0.00001)} # vague priors
       rho ~ dnorm(0,.0001)I(0,)

       # LIKELIHOOD
        for (i in 1 : I)    # loop around elections
        {

      # Multinomial model
         X[i,1:K] ~ dmulti( p[i,1:K] , n[i])
         n[i] <- sum(X[i,])
         for (k in 1:K)    # loop around parties
            {
            p[i,k]        <-  phi[i,k] / sum(phi[i,])
            log(phi[i,k]) <- bias[k] + rho*log(V[i,k])
           }
      }
  }

Data
list( I   = 12,   K   = 3,
      X = structure(.Data = c(
300,  315,  12,
325,  295,  9,
345,  277,  8,
365,  258,  7,
304,  317,  9,
253,  364,  13,
330,  288,  12,
297,  301,  37,
277,  319,  39,
339,  269,  27,
397,  209,  45,
376,  229,  45
), .Dim = c(12, 3)),
V = structure(.Data = c(
.4341994,   .4610934,   .1047072,
.4797144,   .4877813,   .0325043,
.4974225,   .4635792,   .0389983,
.4935235,   .4384425,   .0680339,
.4339797,   .4413256,   .1246946,
.4187992,   .4803503,   .1008505,
.4637579,   .430723,    .1055191,
.3788168,   .3715876,   .2495956,
.3584408,   .3925122,   .249047,
.4387356,   .3693695,   .1918949,
.4242528,   .2757294,   .3000178,
.423, .308, .269), .Dim = c(12, 3))
)
```

Inits

```
list(bias = c(NA, 0, 0),
          rho =  0)
```

king_england_royce_bayes_2.r

```
#
# king_england_royce.r -- King 1990 likelihood estimation, england
# By Royce Carroll and Keith Poole May 2005. Revised July 2006;
November 2009.
#
#
rm(list=ls(all=TRUE))
library(MASS)
#
#  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
#    FUNCTION CALLED BY OPTIM(...) Below
#
# *** Calculate Log-Likelihood ***
# ************************************
fr <- function(rho){
rho_0 <- rho[1]
lambda[1] <- 0.0
lambda[2] <- rho[2]
lambda[3] <- rho[3]
#
i <- 1
logL <- 0.0
while (i <= nelection) {
   rhox <- rho_0
   k <- 1
   logterm <- 0.0
   while (k <= nparty) {
#
#  Calculate constant term
#
       logterm <- logterm + exp(lambda[k] + rhox*log(TT[i,k]))
       k <- k + 1
       }
   j <- 1
   while (j <= nparty){
#
#  Calculate Row
#
       logL <- logL + T[i,j]*(lambda[j] + rhox*log(TT[i,j]) -
log(logterm))
       j <- j + 1
       }
   i <- i + 1
   }
#return(-logL)
#
#  Multiply Likelihood Distribution by Prior 1.0*exp(1.0*rhox) -- When
#   the logs are taken this simply becomes rhox and is subtracted from
#   the Likelihood function.  Note that the priors on the lambdas are
#   normals with variance = 100,000 so they wash out!
#
return(-logL-rhox)
}
```

```
#
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&
#
#  SEATS
#
T <-
matrix(scan("C:/bayes_beach/england_seats.txt",0),ncol=3,byrow=TRUE)
#
#
nrow <- length(T[,1])
ncol <- length(T[1,])
#
#  Set Parameter Values
#
nparty <- ncol
nelection <- nrow
#
#
rho <- NULL
rhomax <- NULL
ncounts <- NULL
xhessian <- NULL
#
lambda <- rep(0,nparty)
#
lambda[1] <- 0.0
lambda[2] <- -0.05
lambda[3] <- -1.6
#
size <- NULL
#
#
#  VOTE SHARES
#
TT <-
matrix(scan("C:/bayes_beach/england_vote_shares.txt",0),ncol=3,byrow=TR
UE)
#
#
# *** Calculate Log-Likelihood ***
#
#rho_0 <- 1.14
rho[1] <- 1.10
# &&&&
rho[2] <- -.05
rho[3] <- -1.6
nparam <- 3
#
# ****************************************
#  DO MAXIMUM LIKELIHOOD MAXIMIZATION HERE
# ****************************************
#
model <- optim(c(0.00,.005,0.0),fr,hessian=TRUE)
#
#  Log-Likelihood (inverse -- optim minimizes!!)
#
```

```
logLmax <- model$value
#
#  Parameter Estimates
#
rhomax <- model$par
#
# convergence an integer code.
#     0 indicates successful convergence.
#
nconverge <- model$convergence
#
# counts
#  A two-element integer vector giving the number of calls to
#    fn and gr respectively.
ncounts <- model$counts
#
xhessian <- model$hessian
#
#  Perform Eigenvalue-Eigenvector Decomposition of Hessian Matrix
#
ev <- eigen(xhessian)
#
#  The Two Lines Below Put the Eigenvalues in a
#    Diagonal Matrix -- The first one creates an
#    identity matrix and the second command puts
#    the singular values on the diagonal
#
Lambda <- diag(nparam)
diag(Lambda) <- ev$val
#
#  Compute U*LAMBDA*U' for check below
#
XX <- ev$vec %*% Lambda %*% t(ev$vec)
#
# Compute Fit of decomposition -- This is just the sum of squared
#  error -- Note that ssesvd should be zero!
#
i <- 0
j <- 0
sseeig <- 0
while (i < nparam) {
  i <- i + 1
  j <- 0
  while (j < nparam) {
     j <- j + 1
     sseeig <- sseeig + (xhessian[i,j] - XX[i,j])**2
  }
}
#
LambdaInv <- diag(nparam)
diag(LambdaInv) <- 1/ev$val
#
#  Compute U*[(LAMBDA)-1]*U' for check below
#
XXInv <- ev$vec %*% LambdaInv %*% t(ev$vec)
#
results <- rep(0,nparam*4)
```

```
dim(results) <- c(nparam,4)
#
results[,1] <- rhomax
results[,2] <- sqrt(diag(XXInv))
results[,3] <- rhomax/sqrt(diag(XXInv))
results[,4] <- pt(-abs(results[,3]),nrow-nparam-1)*2
#
```